PROGRAMAÇÃO WEB

Centro Universitário UniSenac – Campus Pelotas

Escola de Tecnologia da Informação

Prof. Edécio Fernando Iepsen

Vetores

Permitem trabalhar com uma lista de dados

Cada item da lista possui um índice

Tabela 6.1 – Representação dos itens/elementos de um vetor

	produtos
0	Arroz
1	Feijão
2	Iogurte
3	Leite
4	Suco
5	Pão

Para referenciar um item do vetor, devemos indicar seu nome, seguido por um número entre colchetes que aponta para o seu índice. É importante reforçar que o vetor inicia pelo índice 0. Portanto, para obter o primeiro produto inserido no vetor, devemos utilizar: `produtos[0]`.

Métodos para Manipulação dos Vetores

Depois de realizarmos a declaração do vetor, podemos gerenciar a lista com a inclusão e a exclusão de itens a esse vetor. Os principais métodos JavaScript que executam essas tarefas estão indicados na Tabela 6.2.

Tabela 6.2 – Métodos de inclusão e exclusão de itens em vetores

`push()`	Adiciona um elemento ao final do vetor.
`unshift()`	Adiciona um elemento ao início do vetor e desloca os elementos existentes uma posição abaixo.
`pop()`	Remove o último elemento do vetor.
`shift()`	Remove o primeiro elemento do vetor e desloca os elementos existentes uma posição acima.

6.5 Vetores de objetos

Um vetor pode conter uma lista de nomes, como no Exemplo 6.1, ou de números, como no Exemplo 6.2. Além disso, também é possível definir um vetor que contenha uma lista de objetos, com alguns atributos desse objeto. Poderíamos, por exemplo, ter o objeto produto, com os atributos nome, marca e preço. Ou o objeto filme, com os atributos título, gênero e duração.

Definir um vetor de objetos nos permite realizar operações sobre esse vetor, como classificar os seus elementos por um dos seus atributos.

Um vetor de objetos é declarado da mesma forma que um vetor simples. Na inserção de itens no vetor, contudo, devem-se indicar os atributos que o compõem. Observe o script a seguir, que manipula o vetor de objetos carros, com os atributos modelo e preço.

```
<script>
  const carros = []
  carros.push({modelo: "Sandero", preco: 46500})
  carros.push({modelo: "Palio", preco: 37800})
  for (const carro of carros) {
    console.log(`${carro.modelo} - R$: ${carro.preco}`)
  }
</script>
```

```
<script>
  const carros = []
  carros.push({modelo: "Sandero", preco: 46500})
  carros.push({modelo: "Palio", preco: 37800})
  for (const carro of carros) {
    console.log(`${carro.modelo} - R$: ${carro.preco}`)
  }
</script>
```

Nesse script, o vetor é inicialmente declarado. Em seguida, são realizadas duas inclusões de veículos. Atente para a sintaxe que identifica um vetor de objetos: deve-se utilizar as chaves {} para delimitar os atributos e cada atributo deve ser seguido pelos ":" e pelo conteúdo que será atribuído a ele. Na sequência do script, utilizamos o comando `for.. of` para percorrer os elementos do vetor e apresentar o conteúdo de cada um dos seus atributos.

Um detalhe na atribuição de dados de um objeto em JavaScript é que, se o nome da variável for igual ao do atributo, pode-se omitir a atribuição. Observe o exemplo:

```
const carros = []
const modelo = "Fiesta"
const preco = 46800
carros.push({modelo, preco})  // ou carros.push({modelo: modelo, preco: preco})
```

Uso do Map

6.8 Map, Filter e Reduce

Map, filter e reduce são métodos que permitem que operações sobre vetores sejam realizadas de um modo mais eficiente. Vamos apresentar alguns exemplos, para facilitar o entendimento.

Observação: Será empregada a notação das Arrow Functions (funções de seta), que, como veremos no capítulo sobre funções, possuem uma sintaxe mais curta, sem o uso do return em casos de funções com uma única linha.

Começamos com um exemplo sobre o método `map()`:

```
<script>
  const numeros = [10, 13, 20, 8, 15]          // vetor inicial
  const dobros = numeros.map(num => num * 2)   // cada número é obtido e multiplicado
                                               // por 2, criando um novo array
  console.log(dobros.join(", "))               // 20, 26, 40, 16, 30
</script>
```

Na mesma ideia do `for..of` ou do `forEach()` – no sentido de percorrer cada elemento do vetor, o método `map()` cria um novo vetor com o resultado do processamento realizado sobre cada um dos elementos do vetor original. As operações podem ser realizadas também sobre arrays de objetos. Observe o script destacado na Figura 6.6.

LocalStorage

Nos exemplos e exercícios dos capítulos anteriores, principalmente naqueles que manipulavam listas de dados, foi possível perceber que as informações digitadas eram perdidas a cada atualização da página. Neste capítulo, veremos como fazer com que informações se tornem persistentes, ou seja, permaneçam salvas no navegador do usuário. Desse modo, mesmo quando o usuário fechar e abrir o navegador, ou até mesmo reinicializar o computador, sua lista de pacientes, veículos ou itens de um pedido (exemplos já utilizados no livro) poderá ser recuperada, evitando que todos os dados tenham de ser novamente digitados.

Uma das formas de fazer isso no HTML 5 é utilizando o localStorage. Com ele é possível, por exemplo, salvar os dados de um cliente de uma loja, como nome, idade ou clube pelo qual o cliente torce. Como as informações ficam salvas no navegador do cliente, quando ele retornar ao site da loja, é possível obter essas informações e personalizar a página de acordo com as escolhas anteriormente feitas por ele.

Salvar Dados: localStorage.setItem()

Para salvar uma informação no navegador do usuário com o localStorage, devemos utilizar o método `setItem()`. Esse método contém dois parâmetros: chave (nome da variável) e valor (conteúdo da variável). Vamos utilizar um editor online JavaScript, como o site `js.do`, para os nossos primeiros exemplos deste capítulo. Acesse o endereço do site e limpe os códigos do programa de exemplo. Para salvar de forma persistente um conteúdo para a variável `idade`, digite as seguintes linhas:

```
<script>
  localStorage.setItem("idade", 17)
</script>
```

Salvar Dados: localStorage.setItem()

Um detalhe importante sobre esse processo é que os dados salvos pertencem ao navegador e ao domínio que armazenou os dados. Portanto, se você salvar informações utilizando um navegador e depois utilizar qualquer outro para acessar a mesma página, essas informações não serão recuperadas. O mesmo vale para endereços distintos. Se a loja X salvou um conjunto de dados de uma visita de um cliente, a loja Y não terá acesso a essas informações, mesmo que elas utilizem nomes idênticos de variáveis.

Outro detalhe é que o usuário pode remover "acidentalmente" os dados armazenados na Local Storage se limpar o histórico de navegação de seu browser. Portanto, não desenvolva programas para salvar dados relevantes dos usuários com esse recurso.

Recuperar Dados: localStorage.getItem()

Para recuperar, bem como verificar a existência de um dado armazenado no navegador do usuário, devemos utilizar o método `getItem()` com o nome da chave utilizada.

```
const idade = localStorage.getItem("idade")
```

Os dados são salvos como strings. Portanto, cuidado com a execução de cálculos, principalmente de adição, sobre os dados que você armazenou no localStorage.

Remover Dados: localStorage.removeItem()

É importante adicionar em nossos programas uma opção que permita ao usuário remover os dados salvos no localStorage. Para realizar essa tarefa, a linguagem JavaScript dispõe dos métodos `removeItem()` e `clear()`. O método `removeItem()` é utilizado para remover o conteúdo de uma variável salva no domínio da página que o criou. Já o método `clear()`, por sua vez, remove todas as variáveis pertencentes a um domínio e armazenadas em seu navegador.

Por exemplo, se quisermos remover a idade salva no script inicial deste capítulo, devemos retornar ao site `w3schools.com` e digitar o seguinte trecho de código:

```
<script>
  localStorage.removeItem("idade")
</script>
```

https://www.w3schools.com/js/js_json_stringify.asp

Tutorials Exercises Certificates Services Search...

HTML CSS JAVASCRIPT SQL PITÃO JAVA PHP COMO FAZER W3.CSS C C++ C# BOOTSTRAP RE

JSON .stringify()

❮ Anterior

Um uso comum do JSON é trocar dados de/para um servidor web.

Ao enviar dados para um servidor web, os dados precisam ser uma string.

Converta um objeto JavaScript em uma string com `JSON.stringify()`.

Stringificar um objeto JavaScript

Imagine que temos este objeto em JavaScript:

```
const obj = {name: "John", age: 30, city: "New York"};
```

Use a função JavaScript `JSON.stringify()` para convertê-lo em uma string.

```
const myJSON = JSON.stringify(obj);
```

JSON .parse()

❮ Anterior

Um uso comum do JSON é trocar dados de/para um servidor web.

Ao receber dados de um servidor web, os dados são sempre uma string.

Analise os dados com `JSON.parse()`, e os dados se tornarão um objeto JavaScript.

Exemplo - Analisando JSON

Imagine que recebemos este texto de um servidor web:

```
'{"name":"John", "age":30, "city":"New York"}'
```

Use a função JavaScript `JSON.parse()` para converter texto em um objeto JavaScript:

```
const obj = JSON.parse('{"name":"John", "age":30, "city":"New York"}');
```

6.6 Desestruturação e operador Rest/Spread

Um dos acréscimos das novas versões do JavaScript é a possibilidade de atribuir valores às variáveis via desestruturação dos elementos de vetores ou objetos. Podemos começar a demonstração do uso desse recurso no exemplo da seção anterior, onde foi utilizado o vetor de objetos carros, com os atributos modelo e preco. Na rotina de apresentação dos dados, pode-se desestruturar o objeto, conforme o exemplo a seguir:

```
for (const carro of carros) {
  const {modelo, preco} = carro  // "desestrutura" objeto carro em modelo e preco
  console.log(`${modelo} - R$: ${preco}`)
}
```

```
let pacientes = ["João", "Sofia"]
pacientes = ["Ana", ...pacientes]    // acrescenta "Ana" no início do vetor
pacientes = [...pacientes, "Maria"]  // acrescenta "Maria" no final
```

Para esse último exemplo, tenha o cuidado de declarar `pacientes` com `let`, pois ao declarar um array com `const` é possível realizar alterações em seus elementos a partir de métodos como `push()` ou `pop()`, mas não fazer uma reatribuição de valor a ele.

Portanto, os "..." podem servir para "espalhar" os elementos de um array ou objeto (Spread), ou então "juntar" elementos criando um novo array (Rest). No Capítulo 8, voltaremos a discutir sobre o operador Rest na passagem de parâmetros para uma função. O operador Spread também pode ser utilizado para criar uma cópia com os elementos de um vetor e, dessa forma, tem um comportamento semelhante ao método `slice()` – sem parâmetros, discutido anteriormente.

```
const pacientes2 = [...pacientes]  // ou const pacientes2 = pacientes.slice()
```

Exercícios:

- Criar um App para cadastrar e salvar em LocalStorage descrição e preço de uma lista de compras. Para isso:
    - Criar um formulário com os campos descrição e preço
    - Criar uma lista para exibir os itens cadastrados